EXHIBIT 1

UNITED STATES DISTRICT COURT
DISTRICT OF MASSACHUSETTS

TYPEMOCK, LTD., Plaintiff, v. TELERIK INC., Defendant.	No. 1:17-cv-10274-RGS

Declaration of Dr. Benjamin F. Goldberg

Table of Contents

I, Dr. Benjamin F. Goldberg, declare as follows:

I. **Qualifications**

1. I am a tenured Associate Professor in the Department of Computer Science of the Courant Institute of Mathematical Sciences, New York University ("NYU"), in New York, NY. I have held this position since September 1994. From 1987 to 1994, I was an Assistant Professor in the Department of Computer Science at NYU. Since September 2014, I have been the Director of Graduate Studies for the MS programs in the Department of Computer Science, having previously served in that role from September 2009 through August 2012. I served as the Director of Undergraduate Studies for the Department of Computer Science from September 1995 through August 1998 and from September 2003 through August 2006. In addition, I have held a one-year visiting professorship at the Institut National de Recherche en Informatique et en Automatique (INRIA), a national laboratory in France, and was twice appointed to a month-long position as an invited professor at the Ecole Normale Supérieure, a university in Paris.

2. I received my Doctoral degree in Computer Science from Yale University, New Haven, Connecticut in 1988, having previously received Master of Science and Master of Philosophy degrees in Computer Science from Yale in 1984. My undergraduate degree from Williams College in 1982 was a Bachelor of Arts degree with highest honors in Mathematical Sciences.

3. I have taught courses and lectured at the undergraduate and graduate level in programming languages, program design (including debugging and testing), object-oriented programming, virtual machines (including the .NET common language runtime), compilers and other areas related to the technology at issue in this matter.

4. Additional information concerning the computer science courses that I have taught, my professional publications and presentations in the field of computer science, and cases in which I have testified as an expert at trial or deposition in the past five years are set forth in my current Curriculum Vitae, a copy of which is attached as Exhibit A.

II. **Compensation**

5. I am being compensated at a rate of $475 per hour for my work in this case. I will be compensated at the same rate for any deposition I give in this case or if I appear as a witness at a hearing or trial during in this case. My compensation is in no way conditioned on the outcome of this action.

III. **Subject Matter of Opinions**

6. I have been asked by counsel for plaintiff Typemock Ltd. ("Typemock") to form opinions regarding the meaning of certain claim terms found in the asserted claims of US Patents 8,352,923 (the "'923 Patent") and 9,251,041 (the "'041 Patent", collectively the "Asserted Patents"). I have also been asked to form opinions concerning the level of skill of a person of ordinary skill in the art (POSITA) of the Asserted Patents.

7. I understand that Defendant and Defendant's expert, Dr. Orso, have asserted that certain claim elements of the Asserted Patents are "means-plus-function" elements under 35 U.S.C. § 112 ¶ 6 and, further, that such elements are indefinite under 35 U.S.C. § 112 ¶ 2 for failure to disclose corresponding structure in the specification. I have been asked to form opinions on whether each such claim element itself contains sufficient structure to support the element and, further, to address whether the specification of the Asserted Patents sufficiently discloses corresponding structure for the element.

8. I have also been asked to respond to assertions made in the April 20, 2018 Declaration of Professor Alessandro Orso, Ph.D. ("the Orso Declaration") regarding the alleged indefiniteness and interpretation of certain claim terms.

IV. **Materials Considered**

9. The materials that I considered in forming the opinions set forth in this declaration are the following:

- The Asserted Patents and their prosecution history
- The April 20, 2018 Declaration of Professor Alessandro Orso, Ph.D. ("the Orso Declaration")
- Defendant Telerik, Inc.'s Opening Claim Construction Brief
- Plaintiff's Opening Claim Construction Brief

V. **Legal Understandings**

10. I am not an attorney but I have been informed of the following legal standards and principles, which I have used and applied in forming the opinions set forth herein.

11. I am informed that 35 U.S.C. § 112 ¶ 6 expressly provides that a patentee may elect to use means-plus-function language in drafting claims. According to that provision, a claim for a combination may be expressed as a means or step for performing a specified function without the recital of structure, material, or acts in support thereof. I further understand that if the patentee opts to include such a claim, § 112 ¶ 6 provides that such claim shall be construed to cover the corresponding structure, material, or acts described in the specification and equivalents thereof.

12. For claim elements that are determined to be in means-plus-function form, I am informed that the structure disclosed in the specification qualifies as "corresponding structure" if the intrinsic evidence clearly links or associates that structure to the function recited in the claim. In the case of a software-related invention, the "corresponding structure" consists of the algorithms disclosed in the specification, and in the case of a claim to a computer for carrying out the function, the corresponding structure is a computer programmed to implement an algorithm that performs the claimed function. I understand that, for means-plus-function claim elements, the patent must disclose, at least to the satisfaction of one of ordinary skill in the art, enough of an algorithm to provide the necessary structure under § 112, ¶ 6.

13. I am informed that the Court should find a means-plus-function limitation indefinite only if a person of ordinary skill in the art would be unable to recognize the structure in the specification and associate it with the corresponding function in the claim. I am further informed that for computer-implemented procedures, the computer code is not required to be included in the patent specification but, instead, the patentee may express that procedural algorithm in any understandable terms including as a mathematical formula, in prose, as a flow chart, or in any other manner that provides sufficient structure. I understand that the disclosure in the specification need not be exhaustive, since the patent need only disclose sufficient structure for a person of skill in the field to provide an operative software program for the specified function. I further understand that, in considering the definiteness of a means-plus-function claim, the question is not whether one of skill in the art would be capable of implementing a structure to perform the function, but whether that person would understand the written description itself to disclose such a structure.

VI. Definition of a Person of Ordinary Skill in the Art

14. The claimed invention of the Asserted Patents requires that a person practicing the invention have programming experience beyond what is normally found in ordinary applications programming. That is, in my opinion, because practicing the claimed invention involves writing code that interacts with virtual machines (such as the .NET common language runtime), profilers, and/or debuggers, to change the behavior of an executing program, one of skill would have experience in "systems-level" programming – that is going beneath the usual interaction with a piece of software to alter how the software itself is executed by modifying underlying structures in the software system, which are not ordinarily accessible to typical software developers. Thus, in my opinion, a person of ordinary skill in the art would have at least a bachelor's degree in computer science from an accredited academic institution, or the equivalent, and 2-3 years of industry experience in systems-level programming.

VII. Computational apparatus

15. The second limitation of claim 1 of the '923 Patent recites,

> *computational apparatus for at least partially isolating, from within the software application, at least one coupled software component which performs a given function by introducing, prior to execution, code elements for runtime access of application points associated with the at least one coupled software component, wherein at least one code element associated with the at least one coupled software component provides access control between utilizing-utilized software components;*

One of skill reading this limitation would understand that, although the function of the claimed *computational apparatus* is *at least partially isolating, from within the software application, at least one coupled software component which performs a given function*, the structure supporting that function is explicitly specified in the claim element as *by introducing, prior to execution,* *code elements for runtime access of application points associated with the at least one coupled software component, wherein at least one code element associated with the at least one coupled software component provides access control between utilizing-utilized software components*.

16. This structure informs one of skill of the algorithm for performing the isolating function on the at least one software component, namely before the program starts running, introduce code elements at application points for the software component (which is coupled as part of a utilizing-utilized pair of software components), such that at least one of the code elements provides access control between the software component and its partner in the utilizing-utilized pair. The scope of this claim element is clear to a POSITA.

17. Furthermore, there is a wealth of description in the '923 Patent specification of how this algorithm for isolating a software component is implemented in various embodiments of the claimed invention. As an initial matter, the *at least partially isolating* function of the claim is tied, for example, to the '923 Patent specification by the identification at 4:10-12 of "a software isolation system constructed and operative in accordance with certain embodiments of the present invention," where the software isolation system is then discussed in great detail in the subsequent sections describing figures 1-3.

18. Two embodiments are described in the '923 Patent specification that implement the function of *at least partially isolating* by utilizing the algorithm recited in the above claim element, namely *by introducing, prior to execution, code elements for runtime access of application points associated with the at least one coupled software component, wherein at least one code element associated with the at least one coupled software component provides access control between utilizing-utilized software components.* These embodiments are described starting at the top of column 4 and extending through the bottom of column 5 (see, *e.g.*, 4:9-5:61

and figures 1-4). As seen at 4:18-20, the first embodiment performs "insert[ing] a small piece of code 107 that calls the Mock framework 110 which then decides whether to call the original code or to fake the call." In the second embodiment, the algorithm recited in the above claim element for the *at least partially isolating* function is implemented by "changing the metadata tables", as stated at 5:34 of the patent specification. The description of this embodiment at 5:33-47 provides a POSITA with exactly the manner in which the metadata tables are changed to implement the algorithmic structure of the above claim element.

19. The third limitation of claim 1 of the '923 Patent recites,

> *computational apparatus for testing the software application by imposing a fake behavior on the at least one coupled software component, wherein imposing includes removing or replacing an expected behavior of the at least one coupled software component during runtime*

Here, again, one of skill can see that the claim limitation provides both the function of the claimed *computational apparatus* as well as the structure for the apparatus. That is, the function *testing the software application* is explicitly supported by the algorithmic structure, *by imposing a fake behavior on the at least one coupled software component, wherein imposing includes removing or replacing an expected behavior of the at least one coupled software component during runtime.* Although there are, of course, many ways to perform the function of *testing the software application*, this claim element recites only a particular way of doing so and in a manner whose scope is clear to a POSITA.

20. The '923 Patent specification provides a detailed discussion of how the function *testing the software application* is implemented by using the algorithm specified in the above claim element, namely *by imposing a fake behavior on the at least one coupled software component, wherein imposing includes removing or replacing an expected behavior of the at*

least one coupled software component during runtime. In particular, the specification states at 4:25-31,

> The test code 108 calls the Mock framework 110 in order to change the behavior of the production code. Here the test can set up what to fake, how to validate the arguments that are passed, what to return instead of the original code and when to fail the test. The mock framework 110 is responsible for creating mock objects dynamically and for managing the expectations and behavior of all fake calls.

It is clear to one of skill that this test code 108 of the embodiment is performing the testing by, for example, invoking the Mock framework 110 in order to impose the fake behavior on software components by replacing calls to original code – and thus the expected behavior of the original code – with calls to fake code. The majority of the patent specification, such as 5:62 to 13:40 and figures 5-8, is devoted to the numerous ways in which, using the Mock framework, various expected behaviors (e.g., arguments passed to functions, computations performed by functions, and values returned by functions) can be removed or replaced.

21. The first limitation of Claim 9 of the '041 Patent, which is a method claim, recites,

> *at least partially isolating from within the software application, by use of a computational apparatus running a testing application, during runtime, at least one coupled software component which performs a given function by introducing into the software application, prior to execution of the software application, code elements for runtime access of application points associated with the at least one coupled software component, such that at least one of the introduced code elements provides the testing application access between utilizing-utilized software components during runtime*

In this step *a computational apparatus running a testing application* is used to perform the function of *at least partially isolating from within the software application, during runtime, at least one coupled software component which performs a given function.* The claim element itself provides the algorithmic structure to support the function, namely *by introducing into the* *software application, prior to execution of the software application, code elements for runtime access of application points associated with the at least one coupled software component, such that at least one of the introduced code elements provides the testing application access between utilizing-utilized software components during runtime.* In my opinion, it would be clear to one of skill that the algorithmic structure supporting the isolating of the coupled software component (*i.e.* part of a utilizing-utilized pair) to be tested is the introduction of at least one code element before the software application starts running, where the code element provides access by the testing application between the software component and its partner in the utilizing-utilized pair. The scope of this claim element is clear to one of skill in the art of the Asserted Patents.

22. The patent specification provides an extensive description of how this algorithm is implemented in the preferred embodiments. In particular, the portion of the '041 Patent specification at 3:66-5:52 describes two implementations, namely (1) inserting a small piece of code that calls the Mock framework, and (2) changing the metadata tables, that perform the *at least partially isolating* function by implementing the algorithm *introducing into the software application, prior to execution of the software application, code elements for runtime access of application points associated with the at least one coupled software component, such that at least one of the introduced code elements provides the testing application access between utilizing-utilized software components during runtime.*

23. The computational apparatus used in the method of Claim 9 of the '041 Patent is further described in the second limitation of the claim, as follows:

> *testing, by use of the computational apparatus running the testing application, the software application by imposing a fake behavior on the at least one coupled software component, wherein imposing behavior includes removing or replacing an expected behavior of the at least one coupled software component, during runtime, by use of the access provided by the at least one of the introduced code elements*

In this step, the computational apparatus performs the function of *testing … the software application* and the corresponding algorithmic structure explicitly states that such testing is performed *by imposing a fake behavior on the at least one coupled software component, wherein imposing behavior includes removing or replacing an expected behavior of the at least one coupled software component, during runtime, by use of the access provided by the at least one of the introduced code elements*. There are many ways to perform testing of a software application and it is clear to a POSITA that this limitation recites a particular way of doing so.

24. Furthermore, the patent specification provides an extensive description of how the algorithmic structure for this claim element is implemented in the embodiments set forth in the patent. In particular, the passage (already discussed above) at 4:15-21 of the ’041 Patent,

> The test code 108 calls the Mock framework 110 in order to change the behavior of the production code. Here the test can set up what to fake, how to validate the arguments that are passed, what to return instead of the original code and when to fail the test. The mock framework 110 is responsible for creating mock objects dynamically and for managing the expectations and behavior of all fake calls.

and virtually the entirety of the disclosure at 5:53-13:31 and figures 5-8 of the ’041 Patent describe how the function of *testing … the software application* is implemented in the preferred embodiments using the algorithm of *imposing a fake behavior on the at least one coupled software component, wherein imposing behavior includes removing or replacing an expected behavior of the at least one coupled software component, during runtime, by use of the access provided by the at least one of the introduced code elements*.

VIII. <u>Processor</u>

25. The first limitation of claim 1 of the ’041 Patent recites,

> *a first processor functionally associated with a digital memory, which digital memory stores processor executable software testing code adapted to cause one or more second processors to:*

I understand that Defendant and Dr. Orso assert that the term *processor* is indefinite. However, the term *processor* is a well-known term of art in the field and means an electronic device for executing instructions. Furthermore, the '041 Patent explicitly states at 2:57-64,

> Any suitable processor, display and input means may be used to process, display, store and accept information, including computer programs, in accordance with some or all of the teachings of the present invention, such as but not limited to a conventional personal computer processor, workstation or other programmable device or computer or electronic computing device, either general-purpose or specifically constructed, for processing;

Thus, I disagree with Defendant's assertion that the term *processor* in the claims of the Asserted Patents is indefinite. To the extent that Defendant is asserting that there is no support for the *first processor* and the *one or more second processors* of the claim, I disagree. It is clear to a POSITA that, based on a plain reading of the claim language, the second processor of the claim performs the claimed isolating and testing (with substantial support in the patent claims and specification, as discussed above), whereas the first processor simply stores the code that the second processor executes to perform the isolating and testing.

IX. <u>Apparatus for adding access controlling code</u>

26. Claim 32 of the '923 Patent recites,

> *A system according to claim 30 wherein the plurality of software components comprises a set of at least one pairs of utilizing-utilized software components each including a utilized software component and a utilizing software component which utilizes said utilized software component, said apparatus for at least partially isolating comprises apparatus for adding access controlling code between each pair of utilizing-utilized software components, said access controlling code being operative to control access of the utilizing software component to the utilized software component.*

where *said apparatus for at least partially isolating* refers to the computational apparatus recited in the second limitation of claim 30 as follows:

> *computational apparatus for at least partially isolating, from within the software application, at least one coupled software component which performs a given function by introducing, prior to execution, code elements for runtime access and control of application points associated with the at least one coupled software component*

27. As discussed above in relation to the *computational apparatus for at least partially isolating* element of Claim 1, the algorithmic structure in claim 30 corresponding to the *computational apparatus for at least partially isolating at least one coupled software component* is found in the claim itself, namely *introducing, prior to execution, code elements for runtime access and control of application points associated with the at least one coupled software component.* Furthermore (and again as discussed above for the second element of Claim 1), support for the claimed *computational apparatus for at least partially isolating* of Claim 30 is found in the patent specification at, for example, 4:9-5:61 and figures 1-4.

28. Similarly, claim 32 itself refers to programmed computational apparatus (as in the parent claim) that "add[s] access controlling code between each pair of utilizing-utilized software components," which sufficiently expresses to a POSITA, within the claim, how the claimed "adding" is performed. I understand that Defendant and Dr. Orso have also asserted that the term *apparatus for adding access controlling code between each pair of utilizing-utilized software components* of Claim 32 does not have support in the patent specification. I disagree. For example, the "weaver" described in the patent specification, when executing on the processor disclosed and claimed in the patent, is such an apparatus. The following description of the weaver at 4:15-22 of the specification,

> The weaver 104 is responsible for inserting the added hooking code into the production code base 106. In each method of the production code the weaver

> 104 may insert a small piece of code 107 that calls the Mock framework 110 which then decides whether to call the original code or to fake the call. The inserted code 107 can also modify the arguments passed to the production method if required. This is handy for arguments passed by reference.

along with the illustration of the weaver in figure 3 and additional descriptions of the weaver at 4:52-5:30 and elsewhere provide ample algorithmic structure for the claimed *apparatus for adding access controlling code between each pair of utilizing-utilized software components, said access controlling code being operative to control access of the utilizing software component to the utilized software component.*

29. Claim 48 of the '932 patent recites,

> *A system according to claim 30 wherein the plurality of software components comprises a set of at least one pairs of utilizing utilized software components each including a utilized software component and a utilizing software component which utilizes said utilized software component and which includes metadata pointing to the utilized software component, and said apparatus for at least partially isolating comprises apparatus for modifying said meta-data to point to access control code, said access controlling code being operative to control access of the utilizing software component to the utilized software component.*

As in the case of claim 32, claim 48 states that the "apparatus for modifying" modifies the meta-data "to point to access control code," clearly specifying what operation it carries out. I understand that Defendant and Dr. Orso also assert that *apparatus for modifying said meta-data to point to access control code* lacks support in the '923 Patent specification. I disagree. The patent specification describes at 5:33-42 the algorithm for an embodiment in which the metadata tables are modified to control access to a software component (such as a method being called) as follows:

> Another method to isolate code and to insert fake objects is by changing the metadata tables. Each call to a method is defined as 'call <entry in method table>'. Each entry in the method table has the name of the method its type (which is actually an <entry in the type table>) and other information. Each

> entry in the type table has the name of the type and the assembly that it is defined in (which is an <entry in the assembly table>). By switching these entries, for example the assembly of the <type> and its <name> all calls to a method can be redirected to a mocked object.

A POSITA would recognize this algorithm as the structure corresponding to the claimed *apparatus for modifying said meta-data to point to access control code.*

X. <u>Code elements for runtime access</u>

30. Claims 1 and 30 of the ’923 Patent and Claims 1 and 9 of the ’041 Patent all recite *introducing* [] *code elements for runtime access* [*and control*] *of application points associated with the at least one coupled software component.* Defendant and Dr. Orso assert that *code elements for runtime access* is indefinite, but in fact this term would be familiar to a POSITA, particularly in light of the specification of the ’923 and ’041 Patents. One of skill would recognize *code elements* to mean elements of code, namely the instructions and data (such as memory pointers) that are described by the patent specification as being introduced into the software application in order to control access to software components. The term *for runtime access* is also familiar to one of skill in the art as simply meaning to provide access at runtime – that is, while the software application is executing. Thus, in my opinion, a POSITA understand the meaning *of code elements for runtime access* to be the instructions or data that provide access [and control] of the *application points associated with the at least one coupled software component* during execution.

31. This interpretation of *code elements for runtime access* is supported by the patent specification, which describes an embodiment that introduces instructions for providing access [and control] of application points (see, *e.g.*, the ’923 Patent at 4:15-22, 4:36-46, and 10:20-23) and another embodiment that modifies metadata tables, *i.e.* introducing data (*e.g.,* memory

pointers), for providing access [and control] of application points (see, *e.g.,* the ’923 Patent at 5:33-42).

XI. Access controlling code

32. Claim 9 of the ’923 Patent recites,

> *A system according to claim 1 wherein the plurality of software components comprises a set of at least one pairs of utilizing-utilized software components each including a utilized software component and a utilizing software component which utilizes said utilized software component, and wherein said apparatus for at least partially isolating comprises access controlling code external of the software application for anticipating forthcoming utilization of utilized software components by utilizing software components and for selectively preventing said utilization by controlling access of the utilizing software component to the utilized software component.*

and Claim 39 recites,

> *A system according to claim 30 wherein the plurality of software components comprises a set of at least one pairs of utilizing utilized software components each including a utilized software component and a utilizing software component which utilizes said utilized software component, and wherein said apparatus for at least partially isolating comprises access controlling code external of the software application for anticipating forthcoming utilization of utilized software components by utilizing software components and for selectively preventing said utilization by controlling access of the utilizing software component to the utilized software component.*

33. Defendant and Dr. Orso assert that the ’923 Patent specification does not provide sufficient support for *access controlling code external of the software application for anticipating forthcoming utilization of utilized software components by utilizing software components and for selectively preventing said utilization by controlling access of the utilizing software component to the utilized software component*. I disagree. As an initial matter, the term *access controlling code* is readily understood by a POSITA in light of the claim language and the ’923 Patent specification to mean code that controls access to a utilized software component.

34. The patent specification is replete with descriptions of the mock framework as a being code that is *external of the software application* being tested. For example, Figure 1, below, of the '923 Patent clearly shows the mock framework (110) as being external to the production code base (106), namely the software application of the claims.

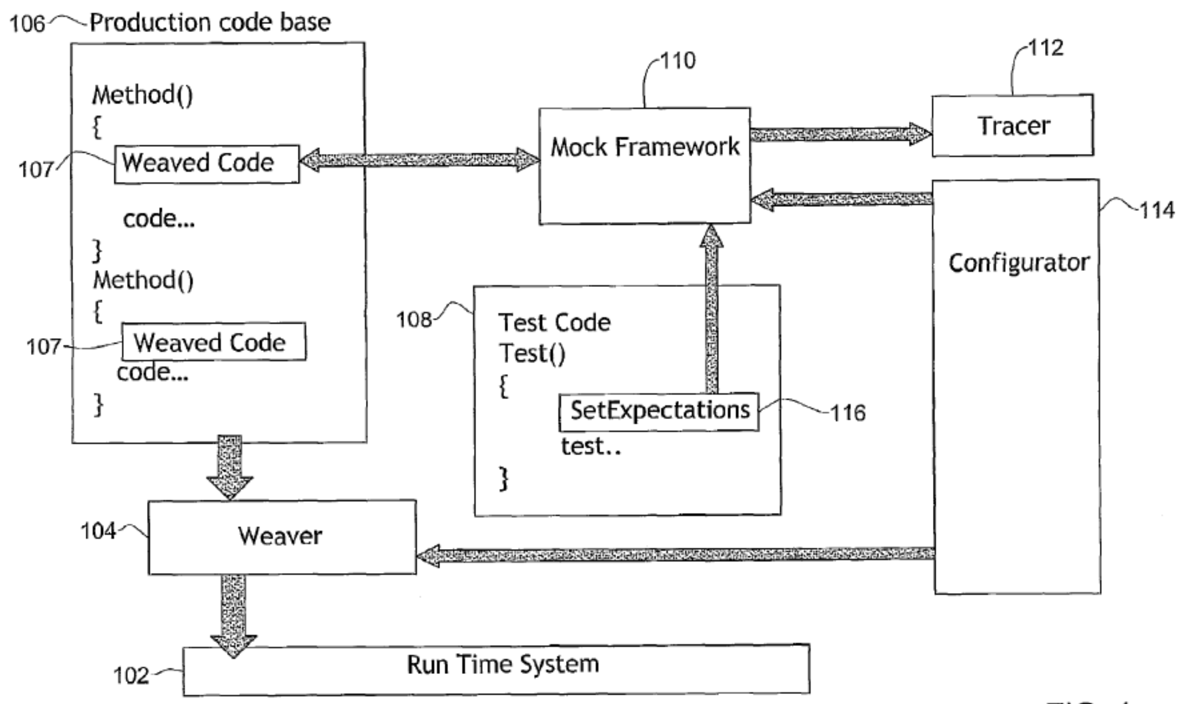

FIG. 1

As another example, the specification at 4:17-20 describes the mock framework as being called from the production code base and returning fake objects to the production code base, thus clearly not part of the production code base:

> certain embodiments of the invention add code that is inserted or weaved **107** into the production code base **106** (FIG. 1) that is being tested. The added code may enable hooking fake or mock objects into the production code by calling the Mock framework **110**. This framework can decide to return a fake object.

35. The '923 Patent specification also provides support for *anticipating forthcoming utilization of utilized software components by utilizing software components*. First, the use of

anticipating in the claim is the usual use of the word (not as a term or art), namely to predict or recognize in advance. Thus, the plain language of *anticipating forthcoming utilization of utilized software components by utilizing software components* means recognizing in advance that a call to a utilized component by a utilizing software component will be coming and preparing in advance to respond to it.

36. The specification describes an implementation of *anticipating forthcoming utilization of utilized software components* in the context of chained calls, such as `Dogs.GetDog("rusty").Tail.Wag( ).Speed(5)` (see the specification at 8:53-65). For example, the specification at 7:3-10 states:

> Chained calls are also supported using Natural Mocks. This allows a chain of calls to be mocked in one statement. The Framework may build the return object of one statement in the chain as an input for the next statement in the chain.

Additional detail regarding the handling of chained calls is provided in the specification at 8:48-9:38. Specifically, that portion of the specification describes that rather than creating a sequence consisting of faking the fetching of a Dog, faking a call to GetDog("rusty"), retrieving a fake tail, enabling the wagging of the fake tail, and then setting the speed of the wagging of the fake tail to 5, each action in this sequence can be anticipated by mocking the behavior for the entire sequence in a single statement, as shown at 8:62-65 in the patent specification.

XII. <u>At least partially isolating</u>

37. Claims 1 and 30 of the '923 Patent and Claim 9 of the '041 Patent recite *at least partially isolating* a software component, which Defendant and Dr. Orso assert is indefinite. However, the meaning of *at least partially* in the claim is the usual meaning of the term, namely the software component is either partly or totally isolated. The specification gives examples of

both cases. One instance in the ’923 Patent specification where a software component is fully isolated is described at 4:17-21 as follows:

> In each method of the production code the weaver **104** may insert a small piece of code **107** that calls the Mock framework **110** which then decides whether to call the original code or to fake the call.

That is, in the case that the Mock framework always decides to fake the call to a method (software component), a POSITA would understand the method to be fully isolated.

38. As an example of partial isolation, the Mock framework may selectively decide at various times to call the original code or to fake the call, as is described in the patent specification at, for example, 4:59-65 and 6:48-53. Another example of partial isolation of a software component is when the original code for a method is called, but the Mock framework changes the parameters that are passed to the method. This is described in the patent specification at, for example, 4:61-5:1. One of skill would understand that all of these cases are examples of *at least partially isolating* a software component.

XIII. <u>Coupled</u>

39. Claims 1 and 30 of the ’923 Patent and Claims 1 and 9 of the ’041 Patent, all recite *coupled software component*. The claims further describe the coupling as being between software components. A POSITA would understand that the term *coupled* has its ordinary meaning, i.e. connected or associated. In software applications, such coupling occurs when one software component calls, invokes, or communicates with another software component. In the case of the Asserted Patents, the specification (and the claim language for some of the claims) describes a *utilizing-utilized* relationship between software components, such as between

methods, in which one method (the *utilizing* method) calls the other method (the *utilized* method).

40. Specific examples of coupled software components in a utilizing-utilized relationship are found in columns 9-11 of the ’923 specification. For example, at 11:60-65 in the ’923 Patent specification, the OpenFile() method calls the file.Open() method, and thus these methods are *coupled* in a utilizing-utilized relationship

XIV. <u>Utilizing-utilized relationship</u>

41. Claims 1, 4, 9, 26, 32, 34, 39, and 48 of the ’923 Patent, and Claims 1 and 9 of the ’041 Patent refer to the *utilizing-utilized relationship*. A POSITA reading the claim language in light of the patent specification would understand that the term *utilizing-utilized relationship* has its ordinary meaning. Specifically, in the context of a *utilizing-utilized relationship* between two software components, the term simply means that one software component uses (utilizes) the other. As discussed above, the ’923 Patent specification, in columns 9-11, provides examples of one method (the utilizing software component) calling another method (the utilized software component, including the OpenFile() method calling the file.Open() method (11:60-65).

XV. <u>An associated behavior inducing message</u>

42. Claim 18 of the ’923 Patent recites,

> *A system according to claim 1 wherein said apparatus for testing is operative to generate a plurality of expectations each of which comprises an identity of an individual component from among the plurality of software components and an associated behavior inducing message inducing said apparatus for at least partially isolating, when said individual component is called, to selectively at least partially isolate, and to impose a fake behavior upon, the individual component.*

43. Defendant and Dr. Orso assert that a POSITA would not understand the meaning of the term *associated behavior inducing message*. In my opinion, however, a POSITA would understand the scope and meaning of *associated behavior inducing message* in the context of claim 18. The term *message* is a term of art that is understood to be a piece of information that is communicated. The rest of the term *associated behavior inducing message* is defined in the claim itself, namely the behavior induced by the message is the causing of *said apparatus for at least partially isolating, when said individual component is called, to selectively at least partially isolate, and to impose a fake behavior upon, the individual component*. As is also clear from the claim language, the *behavior inducing message* is *associated* with *the individual component* in an *expectation* of the claim.

44. The specification discusses expectations in detail at 6:4-64. For example, at 6:16-26, it describes the information associated with a method within an expectation:

> Each Instance Expectation contains a map of Method Expectations 660 that is indexed via the method name. Each method may have the following four lists as shown in FIG. 6:
>
> 1. a default Return Value representing a value to return by default
>
> 2. a queue of return values that should be faked
>
> 3. a queue of conditional values that are used only when the arguments match
>
> 4. a queue of conditional default values are used only when the arguments match

Additionally, the specification discloses that there are hooks that trigger the Mock framework to selectively at least partially isolate a method according to an expectation. The specification states at 6:54-63,

> There are also hooks to call user supplied code when the method is called. As some methods are instance methods, there are ways to tell the Framework what instance to mock. For example, the Framework can be directed to mock all instances, a future instance or to create the mocked instance so that it can be passed to the production code **106** (this may be managed by the Type

> Expectation). Methods can also have conditional expectations. Conditional expectations may fake calls only if the arguments passed are the same as those expected.

XVI. Expected behavior

45. The third limitation of Claim 1 of the '923 Patent recites,

> *computational apparatus for testing the software application by imposing a fake behavior on the at least one coupled software component, wherein imposing includes removing or replacing an expected behavior of the at least one coupled software component during runtime*

Defendant and Dr. Orso assert that the patent claims and specification do not explain what it means for a *coupled software component* to have an *expected behavior*. I disagree. One of skill in the art would understand that virtually the entire disclosure of the '923 Patent concerns testing a software application by removing or replacing the behavior of certain software components, such as methods, in order to impose fake behavior. That is, the behavior expected by a software developer of a software component in the unmodified software application – *i.e.* the *expected behavior* of the software component – is replaced once the software application has been modified by the *computational apparatus for testing*. Thus, in my opinion, a POSITA would recognize that *removing or replacing an expected behavior of the at least one coupled software component* to mean removing or replacing the expected way in which the at least one coupled software component functions or operates.

46. The '923 Patent specification (e.g. at 6:4 to 8:22) provides extensive descriptions of the mechanisms for removing or replacing the expected behavior of a software component in order to fake its behavior. In view of the plain claim language and the specification, a POSITA would certainly understand what is meant by *expected behavior*.

XVII. <u>Reservation of Rights</u>

47. I reserve the right to supplement or amend my opinions in response to opinions expressed by Defendant's experts, or in light of any additional evidence, testimony, discovery or other information that may be provided to me after the date of this declaration.

I declare under penalty of perjury that the foregoing is true and correct.

[illegible]

May 18, 2018
Date

Dr. Benjamin Goldberg